ARCOS – Domain Filter Agent (Retries) – Developer Guide

This guide defines the role, contracts, and reference implementation details for the Domain Filter agent (a.k.a. Filter-Inputs). The Filter agent reduces noise during retries by trimming the Producer's previous output.zip to only files that are implicated in Validator or Post-Processor reports. It also keeps a safe copy of validated/unchanged artifacts and, upon success, reconstructs the full output.zip for a last pass through Validator and Post-Processor before delivery to the user.

1. Rationale & Scope

ARCOS runs in an orchestrated loop with Speculus → Producer → Validator → Post-Processor. On failures, the loop retries with added context. The Filter agent is inserted only on retries to focus the Producer on problem areas, improving iteration speed and signal-to-noise.

Scope:

- Operates only during retry cycles.
- Accepts the previous Producer output.zip and one or more error reports.
- Returns a filtered zip containing only implicated files, plus a manifest.
- Maintains a SafeStore of known-good artifacts for later reconstruction.
- After a successful retry, reassembles a full output.zip by merging fixed files with preserved good files; sends this full package through final Validator and Post-Processor checks.

2. Contracts & Schemas

Messaging follows the ARCOS pattern: Maestro→Agent requests and Agent→Maestro responses. This guide introduces two new schemas for the Filter agent and reuses existing ARCOS schemas for reports.

Referenced (existing) schemas:

- Validator Report: Validator_Report.xsd
- Validator Response wrapper: Validator_Response.xsd
- Post-Processor Report: PostProcessor_Report.xsd
- Producer Response (zip payload): Producer_Response.xsd

New (proposed) schemas:

- Maestro_Filter_Messaging.xsd – request from Maestro to Filter.
- Filter_Response.xsd – response from Filter to Maestro.
- Output_Manifest.xsd – an optional manifest placed inside each output.zip to map domain entities to file paths. Strongly recommended for accurate filtering when Validator diagnostics are entity-centric rather than file-centric.

2.1 Maestro → Filter: Request

Summary: Maestro asks the Filter to trim a previous output against one or more reports.

```
<?xml version='1.0' encoding='UTF-8'?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <xs:element name="FilterRequest">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="ProjectID" type="xs:string"/>
        <xs:element name="PreviousOutputZipBase64" type="xs:string"/>
        <xs:element name="Checksum" type="xs:string"/>
        <xs:element name="Reports">
```

```
          <xs:complexType>
            <xs:sequence>
              <xs:element name="ValidatorReportXML" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
              <xs:element name="PostProcessorReportXML" type="xs:string" minOccurs="0"
maxOccurs="unbounded"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <!-- Optional explicit manifest if not present in the zip -->
        <xs:element name="OutputManifestXML" type="xs:string" minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

2.2 Filter → Maestro: Response

Summary: Filter returns a reduced zip for retry and records a SafeStore location for good files.

```
<?xml version='1.0' encoding='UTF-8'?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <xs:element name="FilterResponse">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="ProjectID" type="xs:string"/>
        <xs:element name="FilteredZipBase64" type="xs:string"/>
        <xs:element name="Checksum" type="xs:string"/>
        <xs:element name="SafeStore" minOccurs="0">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Location" type="xs:anyURI"/>
              <xs:element name="Checksum" type="xs:string" minOccurs="0"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <!-- What was kept/removed for traceability -->
        <xs:element name="FilterSummary" minOccurs="0">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Kept" type="xs:string" maxOccurs="unbounded" minOccurs="0"/>
              <xs:element name="Removed" type="xs:string" maxOccurs="unbounded" minOccurs="0"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

2.3 Output Manifest (inside zip)

A manifest enables deterministic filtering. It maps domain entities (e.g., Bolt/Nut/Washer IDs) to file paths. If absent, the Filter may infer paths using Post-Processor diagnostics and conventional layout rules. Including a manifest is recommended.

```
<?xml version='1.0' encoding='UTF-8'?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <xs:element name="OutputManifest">
    <xs:complexType>
      <xs:sequence>
```

```
        <xs:element name="Item" maxOccurs="unbounded">
          <xs:complexType>
            <xs:attribute name="entity" type="xs:string" use="required"/>  <!-- e.g., Bolt[id=bolt001] -->
            <xs:attribute name="path" type="xs:string" use="required"/>    <!-- e.g., src/bolt.rs -->
            <xs:attribute name="kind" type="xs:string" use="optional"/>    <!-- e.g., code|test|asset -->
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

3. Filtering Algorithm

The Filter agent computes a KeepSet of paths to retain and removes the rest from the previous output zip.
Inputs:
• Previous output.zip (as bytes).
• 1 Validator report or 1 Post-Processor report.
• Optional OutputManifest.xml (either supplied or ***embedded in the zip***).
Steps:
1) Parse reports → collect implicated entities and/or file paths:
  • Validator reports are entity-centric (CRUD or RuleStatus per Part). Extract identifiers.
  • Post-Processor reports often include file paths in diagnostics; extract those.
2) If manifest present, map implicated entities → file paths.
3) Merge file paths from diagnostics and mapped entities into KeepSet.
4) Add always-keep items (e.g., Cargo.toml, build scripts, shared utils, manifest).
5) Create Filtered.zip with only KeepSet files. Store the complement (removed) in SafeStore for later re-assembly.
6) Return Filtered.zip and a FilterSummary (kept/removed) for traceability.

4. Reconstruction Algorithm (Final Pass)

Upon successful Producer retry and passing Validator/Post-Processor checks on the filtered set, reconstruct the final package by merging: (a) Fixed files from the latest Producer output; (b) Previously good files from the SafeStore that were excluded during retries. Then run a final full Validator and Post-Processor pass to ensure the overall solution is valid.

5. Error Handling & Edge Cases

• Checksum mismatches: reject request and ask Maestro to re-send artifacts.
• Empty KeepSet: include minimal project skeleton to allow Producer to rebuild (config, deps, module roots).

6. Security & Isolation

• Never execute code from the zip. Treat artifacts as data.
• Validate and bound XML size before parsing; enforce schema validation on supplied manifest.
• Store SafeStore in a project-scoped, immutable path with content-addressed checksums.
• Preserve provenance metadata (timestamps, checksums) for auditable trails.

7. Reference Pseudocode

```
function filter_zip(previous_zip, reports[], manifest?):
    implicated_entities = set()
    implicated_paths = set()
    for r in reports:
        if r.type == 'validator':
            implicated_entities |= extract_entities_from_validator(r)   # by Part/CRUD/RuleStatus
```

```
        if r.type == 'postprocessor':
            implicated_paths   |= extract_paths_from_diagnostics(r)     # by Diagnostic.File
    if manifest is present:
        for e in implicated_entities:
            implicated_paths |= manifest.lookup_paths(e)
    keep = normalize(add_guard_band(implicated_paths) ∪ always_keep())
    filtered_zip = build_zip(previous_zip, keep)
    removed = all_paths(previous_zip) - keep
    safestore_record = store_removed(previous_zip, removed)
    return { filtered_zip, keep, removed, safestore_record }
```

8. Orchestration Flow (Retries)

1) Validator or Post-Processor returns report to Maestro.
2) Maestro calls Filter with previous output and reports; gets filtered zip + summary.
3) Maestro calls Producer Retry with rules + filtered zip + originating report(s).
4) On success, Filter reassembles full zip and Maestro runs final Validator and Post-Processor.
5) If both pass, Maestro delivers final output.zip to the user.

9. Implementation Checklist

☐ Enforce schema validation on FilterRequest, FilterResponse, and OutputManifest.

☐ Implement parsers for Validator_Report and PostProcessor_Report.

☐ Implement manifest reader/writer; embed manifest into filtered zips.

☐ Maintain SafeStore with content-addressed paths and checksums.

☐ Provide deterministic ordering of zip entries for stable checksums.

☐ Emit a machine-readable FilterSummary (kept/removed) and human summary.

Appendix A – Sample OutputManifest.xml

```
<OutputManifest>
  <Item entity="Bolt[id=bolt001]" path="src/bolt.rs"   kind="code"/>
  <Item entity="Nut[id=nut001]"   path="src/nut.rs"    kind="code"/>
  <Item entity="Washer[id=washer001]" path="src/washer.rs" kind="code"/>
  <Item entity="PackagedKit[id=kit001]" path="src/kits/kit_m12.rs" kind="code"/>
  <Item entity="Project" path="Cargo.toml" kind="config"/>
</OutputManifest>
```